INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS
Int. J. Numer. Meth. Fluids 2011; **67**:1735–1757
Published online 30 November 2010 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/fld.2480

Incompact3d: A powerful tool to tackle turbulence problems with up to $O(10^5)$ computational cores

Sylvain Laizet[1,*,†] and Ning Li[2]

[1]*Turbulence, Mixing and Flow Control Group, Department of Aeronautics, Imperial College London, London SW7 2AZ, U.K.*
[2]*Numerical Algorithms Group (NAG), Wilkinson House, Jordan Hill Road, Oxford OX2 8DR, U.K.*

SUMMARY

Understanding the nature of complex turbulent flows remains one of the most challenging problems in classical physics. Significant progress has been made recently using high performance computing, and computational fluid dynamics is now a credible alternative to experiments and theories in order to understand the rich physics of turbulence. In this paper, we present an efficient numerical tool called *Incompact3d* that can be coupled with massive parallel platforms in order to simulate turbulence problems with as much complexity as possible, using up to $O(10^5)$ computational cores by means of direct numerical simulation (DNS). DNS is the simplest approach conceptually to investigate turbulence, featuring the highest temporal and spatial accuracy and it requires extraordinary powerful resources. This paper is an extension of Laizet *et al*. (*Comput. Fluids* 2010; **39**(3):471–484) where the authors proposed a strategy to run DNS with up to 1024 computational cores. Copyright © 2010 John Wiley & Sons, Ltd.

Received 24 June 2010; Revised 22 September 2010; Accepted 9 October 2010

KEY WORDS: high performance computing; direct numerical simulation; computational fluid dynamics

1. INTRODUCTION

With the recent impressive developments in computer technology, high performance computing (HPC) has entered the reality of petascale computing (systems capable of 10^{15} operations per second) with far-reaching consequences for scientific research. HPC is expected to open the doors to solving highly complex turbulence problems that were until very recently beyond our imagination. Characterized by complex, disorderly motions over a wide range of scales in time and space, turbulence is a grand challenge question that cuts across numerous disciplinary boundaries, from the science of atmospheric phenomena, to the physics of combustion as an energy source for cars or/and jet engines. Many generations of scientists have struggled to understand both the physical essence and the mathematical structure of turbulence.

In the last 50 years, some progress has been made in the understanding of the turbulence problem, thanks to increasingly more complex experiments using advanced and sophisticated techniques, and, of course, thanks to the introduction and widespread use of numerical simulations. The recent unprecedented developments in computer technology has had and will have a strong impact on the turbulence research, especially on three aspects: direct numerical simulation (DNS) of idealized turbulence, increasingly sophisticated engineering models of turbulence and the extraordinary enhancement in the quality and quantity of experimental data achieved thanks to computer storage

*Correspondence to: Sylvain Laizet, Turbulence, Mixing and Flow Control Group, Department of Aeronautics, Imperial College London, London SW7 2AZ, U.K.
†E-mail: sylvain.laizet@gmail.com

and computer post-treatment. John Von Neumann, noted in a 1949 review of turbulence that 'there might be some hope to break the deadlock by extensive, but well-planned, computational efforts'.

From a fundamental point of view, DNS of idealized isotropic, homogeneous turbulence has been revolutionary in its impact on turbulence research because of the possibility to simulate and display the full 3D velocity field without any modelling. One of the most challenging aspects of the turbulence is that the velocity fluctuates over a large range of coupled spatial and temporal scales. If we want to understand the turbulence problem without introducing any bias through numerical methods, it is important to use the most accurate computational approach: DNS. Unfortunately, the computational cost of DNS, even for idealized turbulent flows, is very high, especially when increasing the Reynolds number, as a result of the non-linearity and the non-locality of the Navier–Stokes equations. Furthermore, for flows with relatively complex geometries at the relevant Reynolds numbers (i.e. representative of real situations), the computational resources required by DNS often drastically exceed the capacity of the most powerful massive parallel platforms. In general, because the entire range of length scales appropriate to a given problem cannot be simulated, complex turbulent flow simulations require the introduction of a turbulence model to deal with the smallest scales. Large eddy simulations (LES) and the Reynolds-averaged Navier–Stokes equations (RANS) formulation are the two main techniques used currently to undertake simulations with complex geometries at the relevant Reynolds numbers. However, despite the recent advances in computational fluid dynamics (CFD) methods, the use of sophisticated grids and high-order numerical schemes, the resulting accuracy obtained with these strategies is very often incompatible with the requirements for a detailed analysis of any complex fluid-flow problem and therefore such numerical strategies are not really suitable for a better understanding of the turbulence problem. It is therefore of crucial importance to keep developing numerical strategies for massive parallel platforms in order to simulate turbulence problems with as much complexity as possible without any modelling.

Only very few DNS codes are capable of undertaking massive simulations with several billion mesh nodes on thousands of computational cores. Most of them are simulating idealized homogeneous, isotropic turbulence, using spectral methods with at least periodic boundary conditions in two spatial directions. Indeed, for very simple flow configurations such as homogeneous isotropic turbulence, in terms of accuracy and computational efficiency, the most spectacular gain is obtained using spectral methods [1]. To the knowledge of the authors, the biggest DNS reported in a paper are those performed by [2], where the authors performed DNS of homogeneous isotropic turbulence with 4096^3 mesh nodes. Unfortunately, for fundamental problems in slightly more complex geometry, the full spectral approach is no longer feasible. Note that the spectral element method [3] seems to be a very promising strategy to undertake complex problems with the spectral accuracy. However, using this technique on thousands of computational cores is a challenging task that requires important numerical developments to conciliate accuracy, efficiency and scalability.

The in-house code which forms the basis of this paper is called *Incompact3d* and can combine the versatility of industrial codes with the accuracy of the best academic codes based on spectral methods (the most accurate ones) and can be applied to complex turbulent flows. It is a powerful tool to address rigorously high-resolution simulations of complex fluid-flow problems. *Incompact3d* is already capable of running on 1024 computational cores, with up to 5 billion mesh nodes [4]. However, the HPC landscape is set to change to the point that it would be beneficial to improve the parallel strategy used in the code. Indeed, the domain strategy currently in use in the code is not efficient enough when $O(10^4)$ computational cores are used. As the problem size of our simulations increases, this strong restriction leads to either memory requirement problems or wall clock times longer than desirable for production purposes.

This paper has one main purpose: to describe the formulation of a new domain decomposition strategy and report on the performance of the code on multiple massive parallel platforms with different architectures. It is absolutely clear to us that the efficient and cost-effective exploitation of the current and future high-performance parallel platforms is an essential element for our research and the research of all those at the forefront of fluid mechanics in general and turbulent flows in particular.

DOI: 10.1002/fld

The paper is organized as follows. After a brief description of the numerical methods in Section 2, the new technique used to parallelize the code is described in Section 3. A description of the original Poisson solver is presented in Section 4. The performance of the code is reported in Section 5. Conclusions and future directions are summarized in Section 6.

2. NUMERICAL METHODS AND CURRENT PARALLEL STRATEGY

The main purpose of this paper is not to describe the numerical methods used in *Incompact3d*. For a detailed analysis of the numerical methods, see [5]. However, it could be relevant to recap the main characteristics of the code in order to better understand the proposed parallel strategy presented in this paper.

2.1. General presentation of the code

Incompact3d solves the incompressible Navier–Stokes equations using sixth-order compact schemes for the spatial discretization. For the time integration, different schemes can be used (Adams–Bashforth or Runge–Kutta) depending on the flow configuration. To discretize the parallelepipedal computational domain $L_x \times L_y \times L_z$, a Cartesian mesh of $n_x \times n_y \times n_z$ mesh nodes is used, with the possibility to stretch the mesh in one spatial direction, using a mapping technique proposed by [6, 7]. Inflow/outflow, periodic, free-slip, no-slip can be used in the three spatial directions. The different options for a simulation with *Incompact3d* are presented in Figure 1. To ensure the incompressibility condition, a fractional step method is used, requiring the solution of a Poisson equation for the pressure. Following the technique of [8–10] extended to high-order schemes (that can be adapted on a stretched mesh), the Poisson equation is fully solved in spectral space using Fast Fourier Transform (FFT) routines. It should be emphasized that the Fourier representation can be used, whatever the set of boundary conditions in the three spatial directions, through the relevant use of cosine expansions. Combined with the concept of the modified wave number, this direct (i.e. non-iterative) technique allows the implementation of

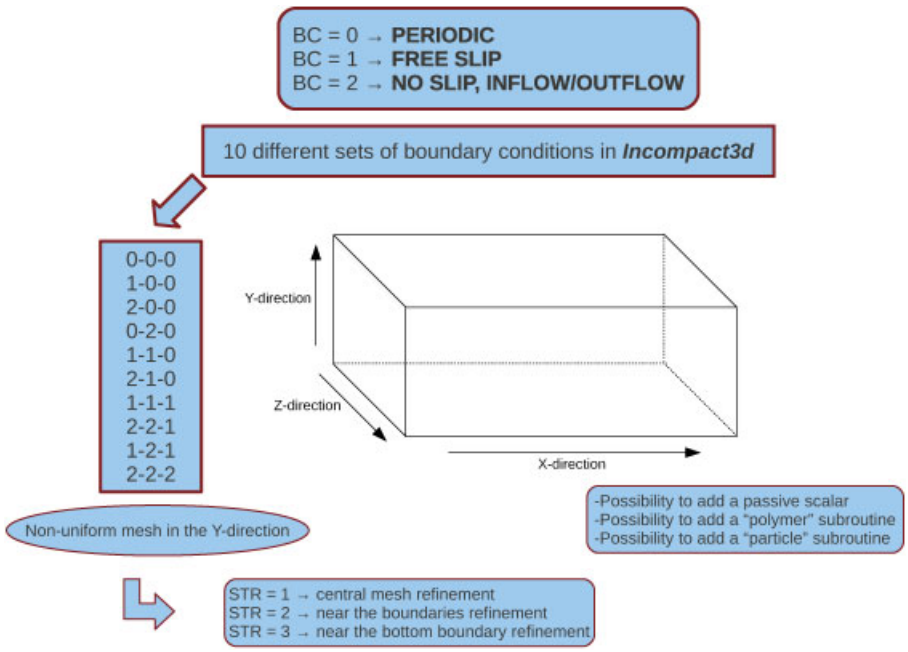

Figure 1. Description of the versatility offered by *Incompact3d*.

DOI: 10.1002/fld

the divergence-free condition up to machine accuracy. A partially staggered mesh is used for the pressure field that is shifted by a half-mesh from the velocity field in each direction. This type of mesh organization leads to a more physically realistic pressure field with insignificant spurious oscillations. Note finally that it is possible to solve an advection–diffusion equation for passive scalars. 'Polymer' and 'Particle' modules have also been recently developed at Imperial College and simulations have been recently conducted with interesting results [11, 12]. More information regarding *Incompact3d*, especially about the Poisson solver, can be found in [5].

2.2. Numerical strategy

Incompact3d is a FORTRAN 90 code that was initially designed for serial processors, then converted to vector processors and more recently converted to parallel platforms [4]. This code is currently widely used for simulations with less than 6 billion mesh nodes on current parallel-architecture platforms. This code is a very attractive tool for DNS and has recently allowed us to study various original flow configurations [13–17].

In order to accurately simulate complex fluid flow problems, three requirements need to be combined by a relevant choice of the numerical method:

- *High accuracy* (*quasi-spectral accuracy*): It is known that the most efficient choice to avoid non-physical effects introduced by the numerical methods is a fully spectral approach. However, such an approach constrains the choice of flow configuration (usually periodic boundary conditions in at least two spatial directions). To allow the use of a wide range of boundary conditions, *Incompact3d* takes advantage of the low computational cost of finite difference schemes. The main idea here is to use operators based on sixth-order compact finite-difference schemes [18] that mimic the behaviour of spectral methods via the so-called 'spectral-like accuracy'. The use of such compact finite-difference schemes on a Cartesian mesh can be seen as a numerical method that is close to the spectral one with only moderate loss of accuracy which is however compensated by the possibility to treat more general boundary conditions than just periodicity such as inflow/outflow boundary conditions, as shown in Figure 1.
- *Ability for complex geometries*: In *Incompact3d*, an immersed boundary method (IBM) is used where the basic principle is to adapt the forcing which replaces and models the effects of the immersed solid body in a way that yields the no-slip condition at the boundary between solid body and fluid. *A priori*, the combination of a high-order scheme with an IBM can be problematic because of the discontinuity in velocity derivatives locally imposed by the artificial forcing term. To the knowledge of the authors, a better accuracy than second order has never been shown in the literature in the framework of IBM. The present code's behaviour in the presence of IBM is consistent with this limitation, with at the best second-order accuracy observed in academic benchmarks [5]. This observation might suggest that the use of high-order schemes is useless, second-order schemes being *a priori* sufficient. This assertion is right formally (in terms of asymptotic convergence) but misleading practically, as shown for instance in [19, 20].
- *Efficiency and portability to massively parallel architectures*: The purpose of this paper is to examine whether this good combination of numerical methods and body modelling can be favourably adapted to massive parallel platforms in order to allow very high-resolution DNS of complex turbulent flows using thousands of computational cores. We will demonstrate that *Incompact3d* is a powerful tool that can undertake DNS with up to $O(10^5)$ computational cores thanks to an efficient 2D domain decomposition.

2.3. The 1D domain decomposition

The first parallel version of *Incompact3d* [4] was developed with several objectives: portability of the code (ability to be run on a wide range of massive parallel platforms), scalability (preservation of the code efficiency when hundreds of computational cores are used) and conservation of the original structure of the code (direct solver for the Poisson equation and sixth-order compact

DOI: 10.1002/fld

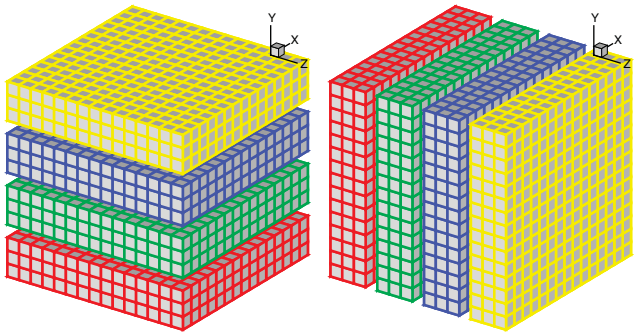

Figure 2. 1D domain decomposition example using four MPI processes: (a) decomposed in Y direction and (b) decomposed in Z direction.

schemes in the three spatial directions). In *Incompact3d*, the spatial differentiation (derivative and interpolation) is ensured by compact sixth-order finite-difference schemes while an explicit scheme (Adams–Bashforth or Runge–Kutta) is used for the time advancement. In practice, the explicit nature of the time discretization does not lead to particular problems for the adaptation to parallel computing. On the contrary, the spatial discrete operators are implicit in the sense that the evaluation of the derivative/interpolation at one node requires to compute all its counterparts along the direction of differentiation. In the context of parallel computing, this property is found to be very penalizing due to the amount of communication exchange that it requires. The present sixth-order schemes require the inversion of a tridiagonal distributed matrix, this generic problem being highly time-consuming because of repetitive communications through the forward and backward dependencies of the computed values node by node.

To ensure the scalability and keep the same structure of the code, the first parallelization strategy was based on a one-dimensional (1D) domain decomposition method: the computational domain is divided into equally sized subdomains in the z direction, ensuring an equal load balance (Figure 2, right decomposition). Each MPI process is then assigned to one subdomain for a simultaneous advancement to the next time level. The algorithm is structured such that no explicit synchronization statements need to be used (same amount of work for each MPI process), which means that the values of all variables are available simultaneously for all subdomains. At this stage, no derivative/interpolation in the z direction can be computed. Therefore, a global transpose operation can be accomplished to get relevant data onto the right MPI process via the use of a second domain decomposition in the y direction (Figure 2, left decomposition). After this operation, calculations can be performed in the z direction for each new slice. The global computation can then be continued after an inverse global transpose operation. These global transpose operations require the parcelling of small blocks of data on each MPI process, labelling these with the address of the MPI-process that requires the information, transferring the data, reading it by receiving MPI process and reconstructing the flow field. It is done with the MPI_ALLTOALL(V) instruction. MPI_ALLTOALL is a collective operation in which all MPI processes send the same amount of data to each other, and receive the same amount of data from each other. However, depending on the total number of mesh nodes, each MPI process may send and receive a different amount of data and provide displacements for the input and output data (for instance, if 65 mesh nodes have to be split into 4 MPI processes, 3 will have to deal with 16 mesh nodes and 1 with 17 mesh nodes). MPI_ALLTOALLV adds this flexibility to MPI_ALLTOALL. It is important to point out that the performance of these two commands is mainly determined by the network hardware and the quality of the MPI library used on each massive parallel platform. Historically, this global transposition technique has been widely used mainly in spectral codes and forms for instance the base of the well-known parallel FFTW library (Fastest Fourier Transform in the West, [21]). Note that the Poisson equation, performed in spectral space, requires 3D FFTs that are performed thanks to the FFTW version 2.1.5, based on the same 1D domain decomposition strategy used in *Incompact3d*.

DOI: 10.1002/fld

2.4. Limitation of the 1D domain decomposition

It has been shown [4] that this strategy can lead to excellent parallel efficiency for the present code with up to 1024 computational cores. Note that this parallelization strategy maintained the original structure of the code since no changes were made in the computation of the spatial differentiations (derivative/interpolation) and in the Poisson solver. Furthermore, as MPI tools and FFTW [21] were used, the portability of the code was maintained and this version of *Incompact3d* is able to run on a wide range of parallel platforms [4].

The main limitation of this first parallel strategy is that it was only possible to use a limited number of computational cores n_c ($n_c < \min(n_y, n_z)$) for a given simulation, the main consequences being a memory requirement problem (each computational core handles too much workload) and/or a computation wall clock time longer than desirable for production purposes (for instance, the 'five billion mesh nodes' DNS performed by [22] took no less than 47 days to get a turbulent state and to collect well-converged statistical data in time). This is a serious limitation as most massive parallel platforms today have more than 10000 cores and some have more than 100000.[‡] As a result, a 2D decomposition strategy, a natural extension to the 1D idea, has been developed for *Incompact3d* and is presented in the next section.

3. 2D DOMAIN DECOMPOSITION

Following the strategy used by CFD codes based on spectral methods which are currently running on thousands of computational cores [2, 24], a new strategy based on a 'pencil' (or 2D) domain decomposition (see Figure 3) is considered in this paper in order to run bigger simulations and/or drastically reduce the wall clock time of our current typical size simulations. This strategy is of course fully compatible with our implicit schemes in space and can be seen as an extension of the current 1D domain decomposition strategy. Note that [24] have observed a 10% extra cost when upgrading their spectral code using a 2D domain decomposition strategy and argued that this is a very small price to pay for the advantage of being able to significantly increase the number of computational cores to run much bigger simulations and/or drastically reduced the wall clock time. In fact, our experience (see Section 5) shows that a properly implemented 2D domain decomposition is not necessarily slower than its 1D counterpart, in particular for large core counts. In [24] is also shown that the scalability of their spectral DNS code (based on the P3DFFT library [25] combined with a 'pencil' strategy) is very good up to 32768 computational cores, which suggests that this strategy should work equally well in *Incompact3D*. At the moment, our biggest simulation with about 5.4 billion mesh nodes ($2785 \times 1392 \times 1392$ mesh nodes) was running on 696 computational cores (with a theoretical limitation of 1392 computational cores) [22]. With the new 'pencil' domain decomposition strategy, we can, in theory, run the same simulation with a theoretical limitation of 1392×1392 computational cores, which will obviously drastically reduce the wall clock time for our simulations.

Figure 3 shows that the same 3D domain as in Figure 2 can be partitioned into two dimensions at a time. From now on, states a, b and c will be referred to as X-pencil, Y-pencil and Z-pencil arrangements, respectively. While a 1D domain decomposition algorithm swaps between two states, a 2D domain decomposition needs to perform global transpose operations among three different states. Different global transpose operations (up to six) can be defined but for *Incompact3d*, we only use four global transpositions: $(a \to b)$, $(b \to c)$, $(c \to b)$ and $(b \to a)$. MPI_ALLTOALL(V) is used to perform the transpositions. It is significantly more complex than the 1D case: there are two separate communicator groups. For a ($P_{\text{row}} \times P_{\text{col}}$) 2D MPI-process grid, P_{row} groups of P_{col} MPI processes need to exchange data among themselves for $(a \leftrightarrow b)$; P_{col} groups of P_{row} MPI processes need to exchange data among themselves for $(b \leftrightarrow c)$. A very important feature here is that when performing the $(a \leftrightarrow b)$ and $(b \leftrightarrow c)$ a given MPI process is not dealing with all the other processes

[‡]The June 2010 TOP500 list [23] shows that 46 of the top 50 systems have more than 10000 cores; 95 of the top 100 systems have more than 5000 cores; the largest IBM BlueGene system Jugene has 294912 cores.

DOI: 10.1002/fld

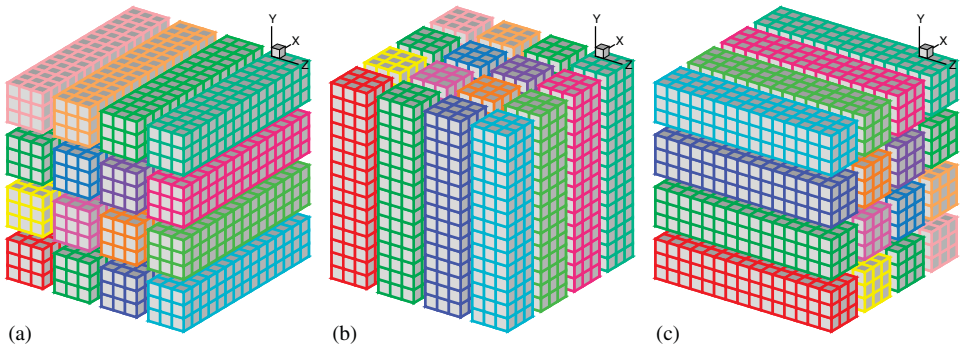

Figure 3. 2D domain decomposition example using a 4×3 MPI processes.

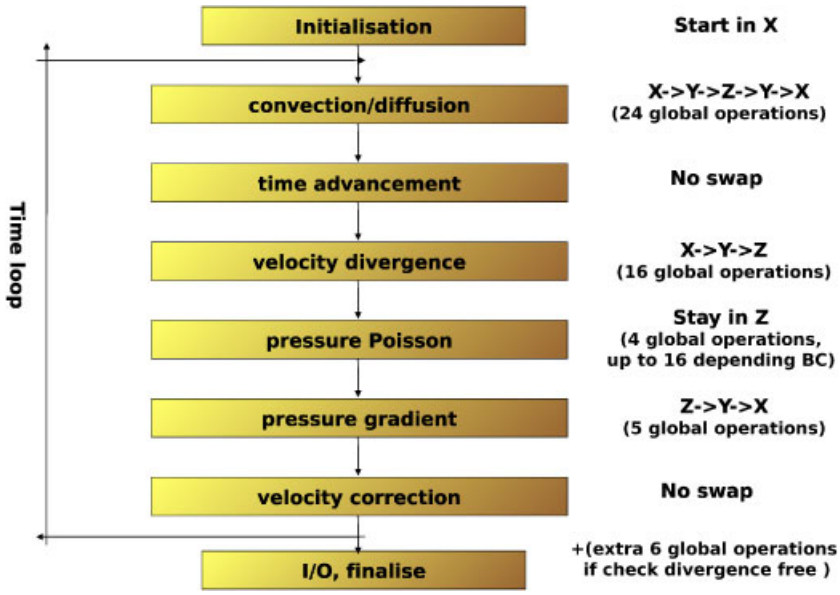

Figure 4. Stucture of *Incompact3d* with the 2D domain decomposition.

(as the name ALLTOALL(V) implies). For instance, for the ($b \rightarrow c$) transposition in Figure 3, the 4 MPI processes of the first line of pencils in b are dealing with the first line of pencils in c. By doing communications in sub-groups, it is possible to drastically reduce the time spent in global communications. Also, this is why there are no direct ($a \leftrightarrow c$) transpositions in *Incompact3d* as they will be very expensive. Finally, it is also worth mentioning that the implementation of the communication routines are very sensitive to the orientations of pencils and their associated memory patterns. Therefore, the packing and unpacking of memory buffers for the MPI library needed to be handled with great care.

It is important to point out again that we did not change the derivative/interpolation subroutines and the Poisson solver as everything is always done in one spatial direction at a time. The derivatives and interpolations in the x direction (y direction, z direction) are performed in X-pencil (Y-pencil, Z-pencil), respectively. The 3D FFTs required by the Poisson solver are also broken down as series of 1D FFTs computed in one direction at a time. In order to reduce the number of global transpose operations, it is necessary to regroup some derivatives/interpolations operations for each spatial direction and compute them at the same time. The structure of the code is presented in Figure 4 in order to demonstrate the management of the pencil swaps. For the Poisson solver in the spectral space, a single division is required and the modified wave numbers combined with the transfer

DOI: 10.1002/fld

functions are all independent of each other. Note that, when performing the 3D FFT forward, we are in Z-pencil in the physical space, then in X-pencil in the spectral space and finally, in Z-pencil again after the 3D FFT backward, in order to reduce the global transpose operations. For the code using tri-periodic boundary conditions, 55 global transpose operations need to be performed at each time step. This number appears to be significant but we will show in the subsequent sections that, despite a large number of global transpose operations, this strategy is suitable for simulations using thousands of computational cores mainly due to its favourable scalability characteristics.

3.1. Shared memory implementation

Modern massive parallel platforms are often based on multi-core processors and cores on the same node often have shared local memory. For the MPI_ALLTOALL(V) type of communication in which each MPI process has to send/receive messages to/from all other MPI processes, we can imagine that when using all the cores of a processor, there is some traffic jam as all the cores on the same physical node compete for their shared network interface. It could become a serious issue on the next generation of massive parallel platforms (such as the recently unveiled Cray XT6 system based on 24-core nodes currently being built in the UK). One possible solution is to create shared send/receive buffers on each node. Then only leaders of the processors participate in the MPI_ALLTOALL(V) communication, resulting in fewer but larger messages passing through the network, hopefully improving the communication performance.[§]

In the base communication library used by *Incompact3D*, a shared-memory implementation has been provided in additional to the standard MPI_ALLTOALL(V) implementation. It utilizes the low-level System V IPC library to allocate shared-memory segments and provides a Fortran wrapper to hide the shared-memory details to applications for better portability. The performance benefit of this technique will be shown later in particular in Figure 10. It is particularly beneficial when a lot of small messages are passed within the system where network latency becomes the performance bottleneck.

4. THE POISSON SOLVER

The originality of *Incompact3d* is that the Poisson equation is solved in spectral space, even if the boundary conditions do not seem well suited for a Fourier representation. It is well known that the treatment of incompressibility is a real difficulty to obtain solutions of the incompressible Navier–Stokes equations. The unavoidable solving of the Poisson equation can be computationally very expensive, especially when high-order schemes are used in combination with iterative techniques. Based on high-order finite difference schemes and expressed in physical space, the inversion of a Poisson equation requires to use sophisticated methods that can be computationally expensive (see [26] for an example of 3D solver of Poisson's equation based on compact schemes). On the contrary, performed in Fourier space, the equivalent operation is cheaper and easy to code with the help of FFT libraries.

In order to drastically reduce any spurious oscillations on the pressure field, the pressure field is staggered by half a mesh with the velocity one. In practice, staggered FFTs are needed, which requires some pre- and post computations in the three spatial directions in physical space and in spectral space [9, 10]. In our distributed solver, it could introduce some extra global transpositions, depending on the boundary conditions. However, we have checked that these extra global transpositions (up to 12 in the worth case) do not affect the good scalability of the code.

[§]Note that we had the chance to run very few simulations on the new XT6 HECToR system based on 24-core processors and we have noticed that the code with the shared memory implementation is 20–30% faster than the classic one.

DOI: 10.1002/fld

4.1. Review of parallel FFT libraries

FFTs have been widely used in the last 50 years and are perhaps the most ubiquitous algorithms used today in CFD. FFT software packages are available everywhere. Almost all hardware vendors propose their own FFT products tuned for a dedicated architecture but there are also many open-source products. Unfortunately, when working on massive parallel systems, the options are very limited.

FFTW [21] is one of the most popular FFT packages available. It is open-source, supporting arbitrary input size, portable and delivers good performance due to its self-tuning design (planning before execution). There are two major versions of FFTW. The version 2.x has a reliable MPI interface to transform distributed data. Note that the old version of *Incompact3d* was based on this version. However, it is based on a 1D decomposition strategy, which, as discussed earlier, limits the scalability of large applications (with more than 1024 computational cores for *Incompact3d*). Its serial performance is also inferior to that of version 3.x, which has benefited from a complete redesign and is much faster than the version 2.x. Unfortunately, the MPI interface of version 3.x is in an unstable alpha stage and has been so for many years. Therefore, there is no reliable way to compute multidimensional FFTs in parallel with this package. For systems based on AMD processors (such as Cray XT HECToR), it is possible to use the FFT routines provided by AMD Core Math Library (ACML)[27] which is specially tuned for AMD architectures. AMD does provide a multi-threading version of the library but there is no distributed support. Thus ACML can only be used to compute the underlying 1D FFTs.

There are several open-source packages available which implement 2D-decomposition based distributed FFTs. For example, Plimpton's parallel FFT package [28] provides a set of C routines to perform 2D and 3D complex-to-complex FFTs together with very flexible data remapping routines for data transpositions. The communications are implemented using MPI_SEND and MPI_IRECV. Takahashi's FFTE package in Fortran [29] contains both serial and distributed version of complex-to-complex FFT routines. It supports transform lengths with small prime factors only and uses MPI_ALLTOALL to transpose evenly distributed data. There is no user callable communication routines.

Finally there is the well-known open-source package P3DFFT [25], which has been widely adopted by scientists doing large-scale simulations in many research areas especially those based on spectral codes. The P3DFFT project was initiated at the San Diego Supercomputer Center by Dmitry Pekurovsky as the foundation of P. K. Yeung's spectral DNS code, which is quite efficient and scales up to 32 768 computational cores [24].

However, initial attempts to adapt this package in *Incompact3d* proved to be impractical. P3DFFT, although suitable for the Poisson solver, targets purely spectral applications, by performing only real-to-complex (r2c) and complex-to-real (c2r) FFTs and its communication routines handling complex data type. There are FFT-specific features (such as the padding of real input for in-place transforms) built in its logic and data structure which are not relevant for finite difference codes that require a more general domain decomposition library to handle global data transpositions. Therefore, we had to develop our own library from scratch, based on a two-layer design with a general-purpose 2D decomposition library as the foundation and a distributed FFT library built on the top. The FFT library is also a general-purpose one, supporting both complex-to-complex (c2c) and real-to-complex/complex-to-real transforms (r2c/c2r) [30].

4.2. FFT engines

The distributed FFT interface only performs data management and communications. The actual computations—all the 1D FFTs—are delegated to a third-party FFT library and are always performed using data in local memory.

To date, *Incompact3d*'s FFT interface supports six different FFT engines: a generic algorithm [31], FFTW (version 3.x) [21], ACML [27], FFTPACK (version 5) [32], Intel MKL [33] and IBM ESSL [34]. There are obviously advantages and disadvantages of using one or the other FFT engines, depending on the hardware and software environment.

DOI: 10.1002/fld

The generic implementation is extended from an algorithm proposed by Glassman [31]. Although not very efficient, it can be used on every massive parallel platform without installing any external library. *FFTPACK* [32] is included here as an FFT engine because it is widely used by legacy applications using elliptic PDE solvers (including Poisson solvers). *FFTW* [21] is the most popular open-source FFT package and is widely used. The FFTW's planning, although very powerful, is not particularly easy to use in this parallel library because it requires proper memory alignment which cannot be guaranteed in Fortran.[¶] All others are highly optimized vendor libraries with the $ACML$ engine [27] specially tuned for AMD processors, the MKL engine [33] tuned for Intel processors and the $ESSL$ engine [34] designed to be efficient on IBM Power-x based architectures (such as BlueGene Jugene). All these engines have their strengths and weaknesses. For example, ACML has a very limited r2c and c2r support; MKL is not really convenient to use as it does not directly accept multi-dimensional arrays as input/output; ESSL only supports a limited choice of FFT length. One objective to implement the distributed FFT library on top of all these FFT engines is to hide all such details (some are purely software engineering issues unrelated to the scientific works) and make the application portable on all kinds of platforms.

5. PERFORMANCE EVALUATION AND SCALABILITY

To investigate the performance of *Incompact3d*, a large number of simulations have been conducted on different platforms, based on different architectures. HECToR in the UK, JADE in France and Jaguar in the US, are based on high-frequency processors with a large amount of memory available per computational core and are limited to 16 384-core simulations for HECToR, 131 072-core for Jaguar and 4096-core simulations for JADE. Jugene in Germany, an IBM BlueGene, is based on low-frequency processors with a relatively small amount of memory available per computational core and is limited to 262 144-core simulations.

The migration on each platform has been achieved with very limited effort due to the excellent portability of the code that required only an MPI library and a Fortran compiler. Note also that for most of the simulations presented in this section, the generic FFT engine had been used (except for the FFT library benchmarking), even if our FFT interface allow us to use a wide range of FFT engines (see Section 4 for more details about our FFT interface).

5.1. Test environment

The HECToR configuration is an integrated system known as 'Rainier', which includes a scalar MPP XT4 system and storage systems. The XT4 comprises 1416 compute blades, each of which has four quad-core processor sockets. This amounts to a total of 22 656 computational cores, each of which acts as a single CPU. The processor is an AMD 2.3 GHz Opteron. Each quad-core socket shares 8 GB of memory. The theoretical peak performance of the system is 208 Tflops. Each quad-core socket controls a Cray SeaStar2 chip router. This has six links which are used to implement a 3D-torus of processors. The point-to-point bandwidth is 2.17 GB/s, and the minimum bi-section bandwidth is 4.1 TB/s (latency around 6 μs). HECToR is located at the University of Edinburgh, in Scotland.

The JADE configuration is an SGI Altix ICE 8200 that comprised 1536 + 1344 compute blades. 1536 compute blades are based on two quad-core processor sockets with 2.8 GHz Intel quad-core E5472 (Hapertown) processors. 1344 compute blades are based on 2 quad-core processor sockets with 2.53 GHz Intel quad-core E5540 (Nehalem) processors. Each quad-core socket shares 32 GB of memory. This amounts to a total of 23 040 computational cores, each of which acts as a single CPU. The theoretical peak performance of the system is 268 Tflops. The bandwidth is 25.6 GB/s among all the compute blades. It is located in Montpellier, at GENCI-CINES (Grand Equipement National de Calcul Intensif—Centre Informatique National de l'Enseignement Supérieur) in France.

[¶]It is necessary to plan every transforms before execution (time consuming) or to plan on globally defined data structures (even temporary work arrays have to be defined globally, which is memory consuming).

DOI: 10.1002/fld

The IBM Blue Gene/P Jugene is an integrated system that includes 72 Racks with 32 nodecards per 32 compute nodes (total 73 728). Each compute node comprises a 4-way SMP 32-bit PowerPC 450 core (850 MHz) for a total of 294 912 computational cores. 2 GB are available per processor. The overall theoretical peak performance is 1 Pflops, with a performance of 825.5 Tflops at the Linpack benchmark. The network is based on a 3D torus with a 12.8 GB/s Ethernet. It is the most powerful platform in Europe with the largest number of computational cores in the world and is located in Julich, Germany.

The Jaguar system consists of an 84 cabinet quad-core Cray XT4 system and 200 upgraded Cray XT5 cabinets, using six-core processors. The XT4 has 8 GB of memory per node while the XT5 has 16 GB per node, giving the users a total of 362 TB of high-speed memory in the combined system. The two systems are connected to the Scalable I/O Network (SION), which links them together and to the Spider file system. The XT5 system has 256 service and I/O nodes providing up to 240 GB/s of bandwidth to SION and 200 GB/s to external networks. The XT4 has 116 service and I/O nodes providing 44 gigabytes per second of bandwidth to SION and 100 GB/s to external networks. With a peak speed of 2.33 Pflops, Jaguar which is located at Oak Ridge Leadership Computing Facility (OLCF) in the US, is the world's fastest supercomputer.

According to the June 2010 ranking issued by the well-known TOP500 website [23], Jaguar, Jugene, JADE, HECToR are, respectively, number 1, 5, 18 and 26 in the ranking of the most powerful massive parallel platforms in the world.

5.2. *FFT library performance*

Large-scale parallel scaling benchmarks of the FFT interface were done on HECToR, Jugene and Jaguar, using problem sizes up to 8192^3 mesh nodes (nearly 550 billion mesh nodes). The timing results presented in Figure 5 are the time spent to compute a pair of forward and backward transforms on a random signal, for complex to complex data (c2c) and for real to complex data (r2c/c2r). The underlying FFT engine is version 4.3 of ACML FFT [27] for all the CRAY systems based on AMD processors. In all cases, the original signals were recovered to machine accuracy after the backward transforms—a good validation for the FFT interface itself. Up to 16 384 cores were used on HECToR, and on Jugene (the larger massive parallel platform at the moment) and Jaguar (the most powerful massive parallel platform at the moment) fewer but larger tests were arranged using up to 131 072 cores.

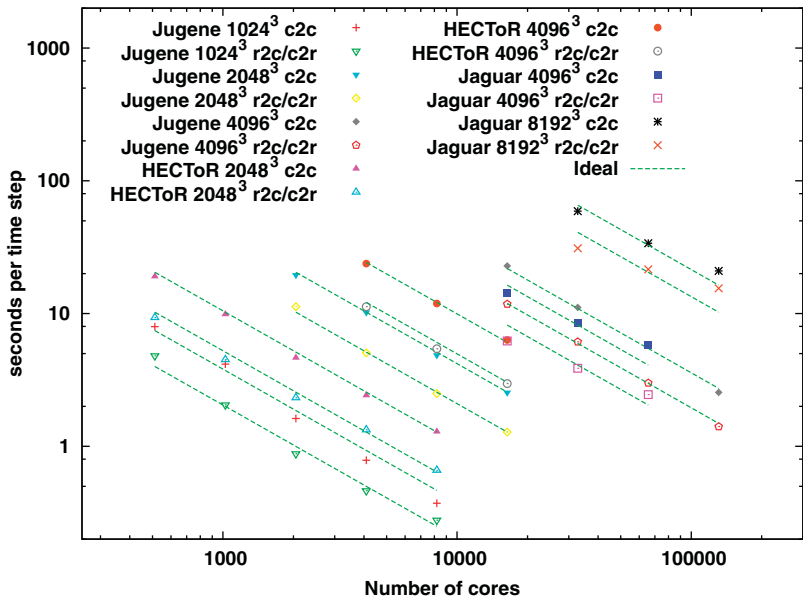

Figure 5. FFT library scaling on HECToR, Jugene and Jaguar.

DOI: 10.1002/fld

Table I. Poisson solver performance on HECToR.

Size	Cores	B.C. 0-0-0 s/time step	B.C. 1/2-0-0 s/time step	B.C. 0-1/2-0 s/time step	B.C. 1/2-1/2-0/1/2 s/time step
1024^3	128	4.81	7.38	6.81	8.23
2048^3	1024	6.26	10.38	8.86	11.56
8192^3	8196	7.59	14.41	12.63	16.31

It can be seen that the FFT interface scales almost perfectly on HECToR for all the tests done. As expected, the r2c/c2r transforms are nearly twice as fast as the corresponding c2c ones. On Jaguar, the scaling is also close to the ideal one for larger core counts and the efficiency is 81% for the largest test. For one particular problem size, 16384-core simulation on 4096^3 mesh nodes, Jaguar took twice as much time to run than HECToR. This is not a surprise. Indeed, while HECToR has quad-core processors at the moment, Jaguar has two six-core chips built on each node. The problems set up for these benchmarks really prefer a power-of-2 core count to run efficiently as the communication network can be used in a balanced way.[‖] On Jugene, the scaling is also seen to be very good.[**]

5.3. Influence of the boundary conditions

As already stated previously, it is possible to use all kinds of boundary conditions for the velocity despite the use of a fully spectral Poisson solver (see Figure 1). Because the pressure field is staggered by a half mesh with the velocity field, passing from/to physical space to/from spectral space require some pre and post computations (see [9] and [5] for the basic principle of these computations). The algorithm involves the following steps:

- Pre-processing in physical space (with eventually global transpose operations)
- 3D forward FFT
- Pre-processing in spectral space (with eventually global transpose operations)
- Solving the Poisson by a division of the modified wave numbers combined with the associated transfer functions
- Post-processing in spectral space (with eventually global transpose operations)
- 3D inverse FFT
- Post-processing in physical space (with eventually global transpose operations).

The forward and backward transforms are standard FFTs (even for data sets with non-periodic boundary conditions), which are passed to the FFT library. Depending on the set of boundary conditions, some of the pre- or post processing steps may be optional (for instance when periodic boundary conditions are applied in the three spatial directions, see Table I). Without giving any mathematical details, the pre- and post processing involves operations which evaluate modified wave numbers and package a Discrete Cosine Transforms into a suitable form so that standard FFT routines can be used efficiently. More details can be found in [5].

These pre- and post processing can be either local (meaning that operations can be done regardless of the parallel distribution of data), or global (meaning that calculations are only possible when data sets involved are all in local memory, i.e. the operations have to be done in a particular pencil-orientation). Fortunately, for the global case, whenever the data required are not available, the global transposition routines provided by the decomposition library can be used to redistribute the data.

[‖]For example, a communicator with 4 members always sits on the same physical chip on HECToR while this is not guaranteed on either Jaguar (or the 24-core HECToR XT6 system that is currently being built).

[**]The apparent super-linear scaling achieved for the 1024^3 mesh case is due to the switch to the TORUS network topology at 1024 cores.

DOI: 10.1002/fld

The number of global transpositions required is dependent on the boundary conditions, and the performance of the FFT library is summarized in Table I. Here, the boundary type 0, 1 and 2 correspond to those defined in Figure 1. In the worst case $1/2-1/2-0/1/2$, 12 additional transpositions are required for the pre- and post processing whereas the 3D FFTs themselves (one forward and backward pair) contain only four global transpositions. In that case, 66 global transpose operations are performed per time step in *Incompact3d*. As expected, there is a factor of up to 2 between a 0-0-0 simulation and a 1/2-1/2-0/1/2 one for wall clock time. The extra number of global transpose operations appears to be high. However, these extra operations do not affect the good scalability of the code. Indeed, for each time step (or each sub time step), we only use the FFT library once. Furthermore, because FFT is such computationally intensive algorithm, the actual benchmark results on three large problem sizes (1024^3, 2048^3 and 4096^3) show that the communication cost is only a small proportion of the total wall clock time.

5.4. Scalability

In the context of HPC, there are two common notions of scalability. The first notion is strong scaling, which is defined as how the wall clock time of a simulation varies with the number of computational cores for a global fixed problem size: ideally, the problem will run twice as fast when the number of computational cores is doubled. The second is weak scaling, which is defined as how the wall clock time varies with the number of computational cores for a fixed problem size per computational core: ideally, when both the size of the problem and the number of computational cores are doubled, the wall clock time remains constant.

Usually, the strong scaling is evaluated with the strong speedup that represents the gain with respect to a sequential calculation. However, this criterion does not give always satisfying information. First, a good speedup can be related to poor sequential performance (for example, if a computational core runs slowly with respect to its communication network). Second, the sequential time is usually not known, mainly due to limited memory resources. Except for the vector platform, the typical memory size available for one computational core ranges from 0.5 to 32 Gb. As a consequence, in most of our configurations, the simulation cannot be run on a scalar platform with a single computational core and no indication about the real speedup can therefore be obtained. A definition of the speedup more often used, is related to the time ratio between a calculation with N_{used} computational cores and N_{min} computational cores, where N_{min} is the smallest number of computational cores that can be used to run the simulation using the maximum memory available on a given massive parallel platform.

In order to measure the scalability but also to test the good portability of the new version of *Incompact3d*, several simulations were performed with different number of mesh nodes (from 134 million mesh nodes to 68.7 billion mesh nodes) and computational cores (up to 262 144), depending on the memory capacity of each platform. The simulations are presented in Table II. Note finally that all the simulations presented in this section have been performed with periodic boundary conditions in the three spatial directions and with the generic FFT engine.

Table II. Parameters of the simulations performed for the scalability investigations.

(n_x, n_y, n_z)	Computational cores	Platform
(512, 512, 512)	1024	JADE, HECToR, Jugene
(1024, 256, 256)	$32 \rightarrow 512$	HECToR
(2048, 512, 512)	$128 \rightarrow 4096$	JADE
(2048, 512, 512)	$128 \rightarrow 16\,384$	HECToR
(2048, 512, 512)	$1024 \rightarrow 131\,072$	Jugene
(2048, 2048, 2048)	$1024 \rightarrow 16\,384$	HECToR
(2048, 2048, 2048)	$2048 \rightarrow 131\,072$	Jugene
(4096, 4096, 2048)	$131\,072 \rightarrow 262\,144$	Jugene
(4096, 4096, 4096)	$8192 \rightarrow 16\,384$	HECToR

DOI: 10.1002/fld

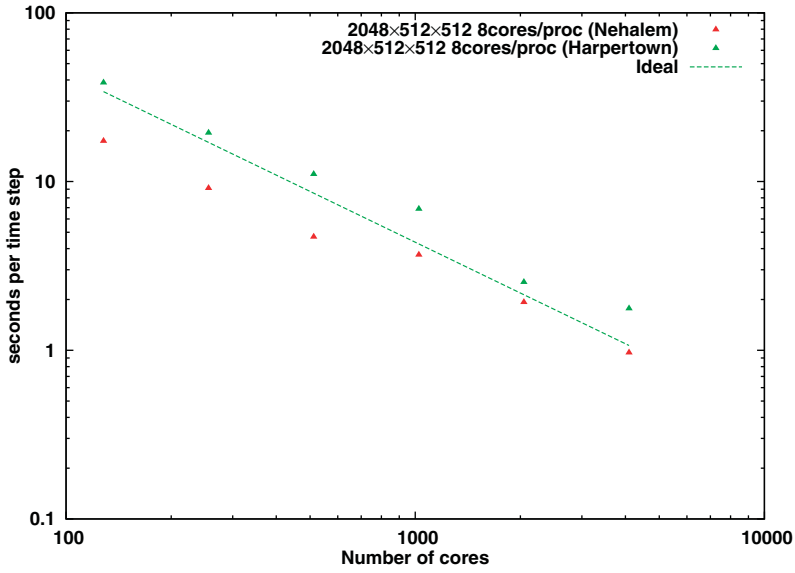

Figure 6. Strong scaling on JADE.

As we only had a limited access to JADE, we were able to run only a small number of simulations on that platform. For a simulation with about 537 million mesh nodes, *Incompact3d* shows an excellent scalability with up to 4096 computational cores in Figure 6. Despite the large amount of communication exchanges needed for the 2D domain decomposition, the scalability remains quite close to the ideal one. The super-linearity observed with more than 2048 computational cores is due to the network performance. We have observed that when using 2048 computational cores or more on JADE (with the Hapertown processors, the network with the Nehalem processors being more recent), the communications can be performed faster thanks to the hypercube structure of the Infiniband network. Note that a similar behaviour can be observed on Jugene in Figure 5 when using 2048 computational cores or more. As expected, there is a factor of more than 2 between the simulations performed with the Nehalem and Hapertown processors. There is more memory and network available for the Nehalem processors (newer and faster processors) so that there is less traffic jam among the computational cores inside each processor and therefore the code can run faster.

This important point about traffic jam is also exhibited on HECToR where we performed a $1024 \times 256 \times 256$ mesh nodes simulation with 1, 2 and 4 cores per processor, as shown in Figure 7. Using less computational cores per processors means that there is more network bandwidth available for each MPI process inside each processor so that the global transpositions can be done much faster. For instance, for 512 computational cores, this simulation is more than twice faster if performed with only 1 computational core per processor rather than 4.

The very good behaviour of *Incompact3d* is confirmed on HECToR where we were able to run a large number of simulations with up to 16 384 computational cores, with up to 68 billion mesh nodes. We also tried to evaluate the limit of the new domain decomposition strategy for each platform with a relatively small size problem and a large number of computational cores. For the $2048 \times 512 \times 512$ simulation, the scalability is excellent with up to 8192 computational cores and is really poor when using 16 384 cores. This is the result of assigning a too small workload to each process, the network latency becoming dominant eventually so that the scalability cannot be improved. However, for a simulation with four times more mesh nodes (2048^3 mesh nodes), the scalability is excellent up to 16 384 computational cores. Furthermore, it is important to note that, it was possible to achieve a 92% efficiency for the simulation with 68 billion mesh nodes when using 8192 and 16 384 computational cores. The last configuration represents 72% of the system capacity and is currently not relevant for production purpose: it is obviously not realistic

DOI: 10.1002/fld

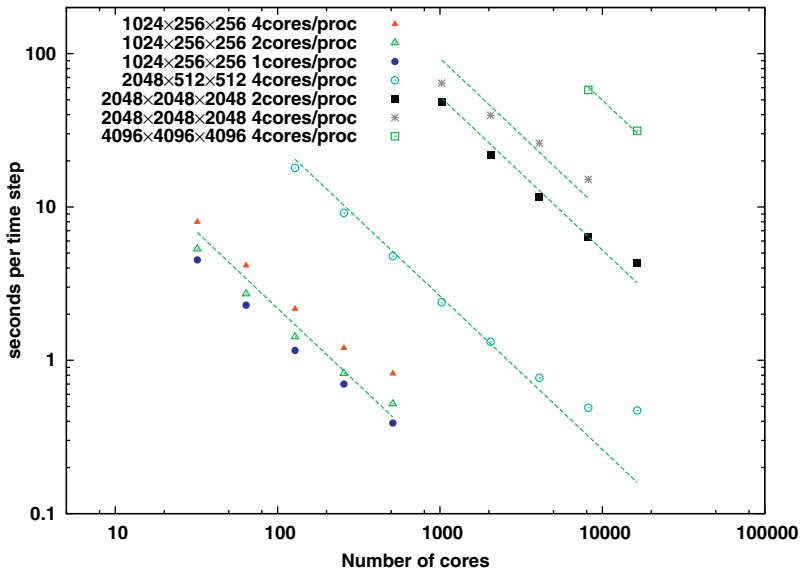

Figure 7. Strong scaling on HECToR.

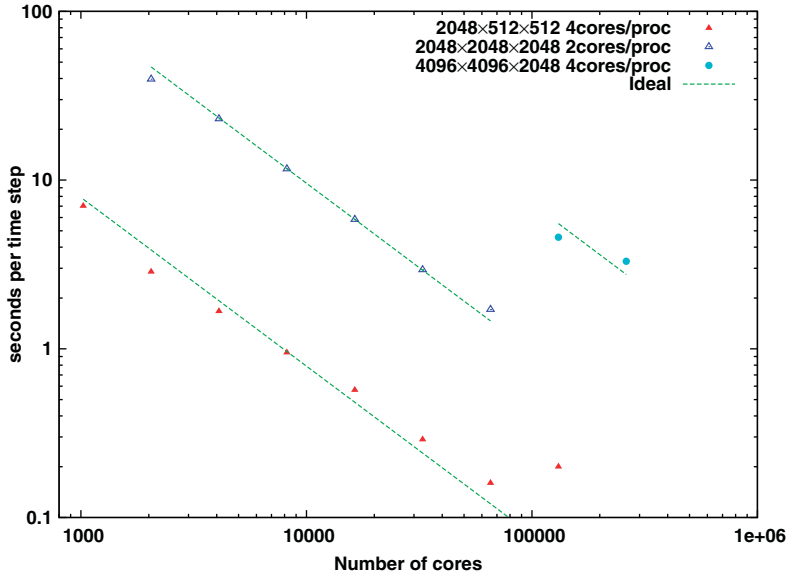

Figure 8. Strong scaling on Jugene.

to undertake production simulations with such a number of computational cores on HECToR for obvious practical reasons but this extreme test shows the impressive behaviour of the code when a large amount of global operations is needed.

We also had the chance to investigate the properties of the code on Jugene which has a different architecture by comparison with JADE and HECToR. Indeed, Jugene is based on low frequency processors with a small amount of memory available per computational core. We have been able to check the limit of the new version of *Incompact3d* with simulations performed with up to 262 144 computational cores. The results are presented in Figure 8. Again, we have observed a very good scaling behaviour for the code in general. For our current typical size production simulations ($2048 \times 512 \times 512$ mesh nodes), we can observe a good scaling up to 65 536 computational cores. The same limitation observed with HECToR with 16 384 computational cores is recovered here but with 131 072 cores computational cores. Note that for this simulation, it was only possible to

DOI: 10.1002/fld

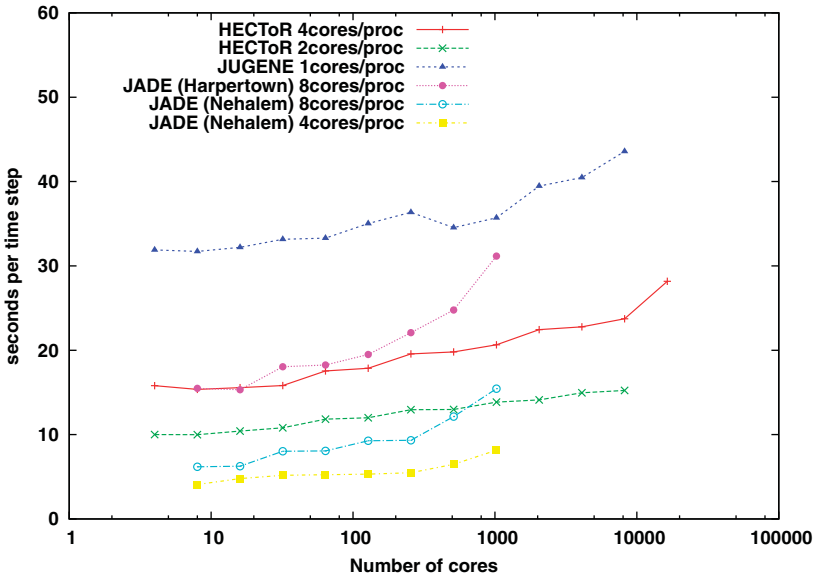

Figure 9. Weak scaling with fixed size problem of 4 194 304 mesh nodes ($4.2M$) per computational core.

use a maximum number of 512 computational cores with the old 1D domain decomposition. It is now possible to drastically reduce the wall clock time for our current production simulations: a 3-month simulation with the old code can easily become a 2-day simulation with the new domain decomposition strategy. For bigger size simulations, we can observe a decent scalability with up to 262 144 computational cores. For such a number of computational cores, the number of communications (that are performed at the same time) is huge but the size of them is very small. This is probably why the scalability is not perfect when hundreds of thousands of computational cores are used.

As already said, weak scaling indicates how the restitution time varies with computational cores count with a fixed problem size per computational core. It is also a very good way to compare the performance of the different massive parallel platforms. In Figure 9, some weak scaling results, obtained on JADE, HECToR and Jugene for the code are presented. For this study, we have fixed a size of 4 198 400 mesh nodes per computational core, and we have increased the number of computational cores, depending on the architecture of the platform. We have also investigated the effect of playing with the number of MPI processes per processors. Even if the number of communications is increasing when increasing the number of computational cores, the wall clock time is globally increasing very slowly because all the communications are performed at the same time. As expected, the fastest results are obtained with Jade (Nehalem, that have a very good cache performance) using only four cores per processors and the slowest results are obtained with Jugene, which is based on very slow frequency processors. For instance, for 512 computational cores, Jade is capable of running the simulation using only 6.47 s/time step while Jugene is using 34.54 s/time step and HECToR 19.80 s/time step (4 cores/proc) and 12.98 s/time step (2 cores/proc). Note that we also checked the performance of the old code based on a 1D domain decomposition on HECToR and obtained a performance of 22.61 s/time step for the same number (512) of computational cores. It proves that the new 2D domain decomposition is more efficient than the old one when using the same number of computational cores, mainly because the global transpose operations are not really global, as explained in Section 3.

As the frequency of the processors is well known, these simulations can eventually exhibit the difference in the performance of the different networks. For instance, Jugene is based on 0.85 GHz computational cores and HECToR on 2.3 GHz ones (HECToR computational cores are 2.7 times faster). Assuming that Jugene is twice faster when using 1 core per processor rather than 4 cores

DOI: 10.1002/fld

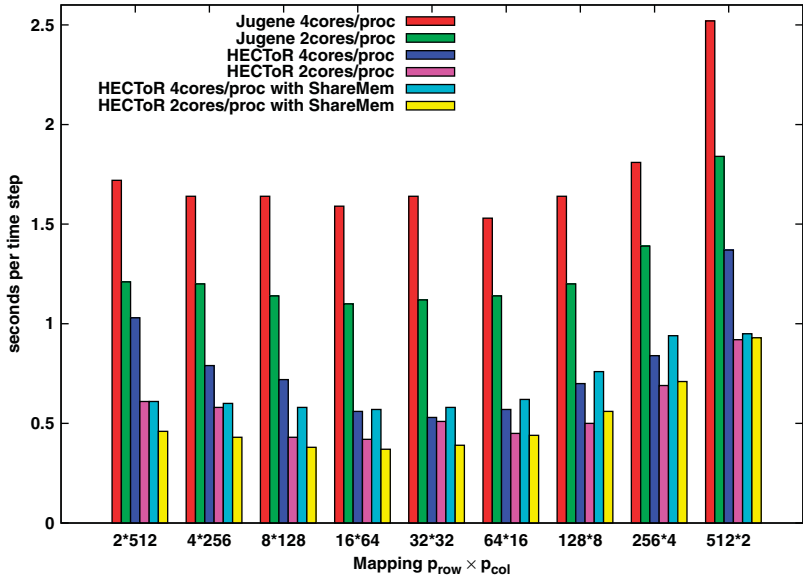

Figure 10. Mapping for $512 \times 512 \times 512$ mesh nodes with 1024 cores.

per processor and that $34.54/19.80 = 1.74$, we can say that Jugene is about 3.48 time slower than HECToR when using 4 cores per processor. As we have a ratio of 2.7 between the frequency of the processors, we can conclude that the performance of the Jugene network is about 30% slower than the HECToR network, at least for *Incompact3d*. Therefore, if we want to have the same wall clock time on HECToR and on Jugene for a given simulation, we have to use twice more computational cores with 2 cores per processor or use four times more computational cores with 4 cores per processor on Jugene.

In Figure 10, we present the influence of the shape of the 2D MPI process grid $P_{\text{row}} \times P_{\text{col}}$ for $512 \times 512 \times 512$ mesh nodes with 1024 computational cores. In Figure 11, we present the influence of the shape of the 2D MPI process grid $P_{\text{row}} \times P_{\text{col}}$ for $2048 \times 2048 \times 2048$ mesh nodes with 2048 computational cores on HECToR only, with the share memory implementation (see Section 3.1). Depending on the parallel platform architecture, in particular the network layout, some 2D grid options deliver much better performance than others. It is, therefore, of crucial importance to test these issues before running large production simulations. It can be seen that, subject to constraint $\max(P_{\text{row}}, P_{\text{col}}) < \min(nx, ny, nz)$:

- The code is always faster when using only few cores per processor, because of improved network bandwidth and memory bandwidth.
- As expected, due to its low clock-speed processors, Jugene is much slower than HECToR even when 2 cores per processor are used.
- When $P_{\text{row}} \gg P_{\text{col}}$, the code is very slow, and this conclusion is independent of the system. This is related to the memory cache behaviour.[††]
- When $P_{\text{row}} \leqslant P_{\text{col}}$, the code is faster than when $P_{\text{row}} > P_{\text{col}}$.
- It seems that the fastest option for the 2D grid mapping is obtained when $P_{\text{row}} \simeq P_{\text{col}}$.
- The code with the shared memory implementation is slightly faster when $P_{\text{row}} \simeq P_{\text{col}}$.

It is clear that the choice for P_{row} and for P_{col} for a given problem is of crucial importance in order to reduce the wall clock time of a simulation. Note however, that these conclusions may not be representative for a large number of computational cores or for bigger size problems. In fact the behaviour is also highly dependent on the network hardware, the time-varying system workload

[††]For distributed 3D arrays, the first dimension, often the inner-most loop, is of size n_x/P_{row}. A smaller P_{row} translates into better memory striding patterns.

DOI: 10.1002/fld

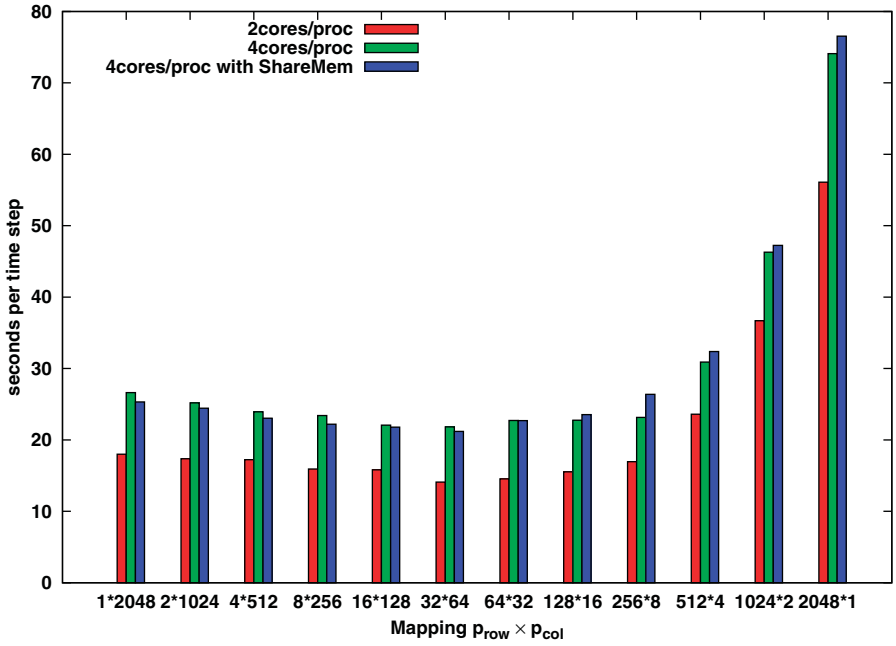

Figure 11. Mapping for $2048 \times 2048 \times 2048$ mesh nodes with 2048 cores.

and the size and shape of the global mesh of the system, among other factors. An auto-tuning algorithm is included in the code in order to allow the best 2D MPI process grid $P_{\text{row}} \times P_{\text{col}}$ to be determined at runtime for production simulations.

5.5. Communication cost

In order to evaluate the cost of the 2D domain decomposition strategy, several simulations have been conducted on JADE, HECToR and Jugene using profiling software: CrayPat and Cray Apprendice [35] on HECToR and open-source package Scalasca [36] on JADE, HECToR and Jugene.

In *Incompact3d*, the cost of the communication depends on both the number of data exchanged among the MPI processes and the size of each data exchanged. Naturally, they are linked to the number of computational cores (in this paper, we always have one MPI process per computational core). Indeed, the number of data exchanged increases with the number of MPI processes whereas the size of each data exchanged decreases. Table III shows the time cost for the main steps in *Incompact3d* in the time advancement loop and also the global time cost of the communications for a $2048 \times 512 \times 512$ mesh nodes problem.

As expected, three different kinds of behaviour can be observed, strongly depending on the architecture of the platform. Owing to the 2D domain decomposition strategy, most of the time is spent in performing the different global transpose operations, especially on JADE where the processors are very powerful. Therefore, up to 88% can be spent in communication, even if it does not affect the scalability of the code. On the contrary, less than 50% of the global time is spent in communication on Jugene. Indeed, because the frequency of the processors is very slow, more time is spent in the computation of the derivatives/interpolations. These trends clearly show that it is not really relevant to think that the scaling of a code depends only on the communication time, as the scaling for this problem size is excellent on the three platforms. It should be noted that, as the number of mesh nodes remains constant, the global transpose operations always concern the same total amount of data. Therefore, it is natural to expect almost the same percentage for the communication cost when increasing the number of computational cores, at least until a certain limit where the data exchanged are really too small for efficient calculation and/or communication. Finally, it is important to say that the 2D domain decomposition is almost as much expensive as the previous 1D domain decomposition. Laizet *et al.* [4] reported that 42% is spent in communications

DOI: 10.1002/fld

Table III. Detailed report of the cost for the main steps in a time loop for *Incompact3d*, for a $2048 \times 512 \times 512$ simulations, with different numbers of computational cores, from 128 to 1024 on JADE, from 128 to 8196 on HECToR and from 1024 to 8196 on Jugene.

Cores	Conv-Diff. (%)	Poisson (%)	GradP (%)	Div(∗2) (%)	Com (%)	System
128	44.04	15.75	11.97	27.45	75.35	JADE
128	41.52	20.98	11.44	24.67	51.80	HECToR
256	47.30	14.39	10.39	27.30	69.93	JADE
256	41.77	20.13	11.37	25.08	51.76	HECToR
512	52.47	11.71	9.53	25.71	83.74	JADE
512	41.38	19.12	11.04	24.61	53.98	HECToR
1024	49.51	15.17	9.33	25.39	87.93	JADE
1024	43.20	18.69	11.18	24.23	56.42	HECToR
1024	38.34	27.07	9.68	22.52	45.63	Jugene
2048	42.33	16.33	9.89	23.02	58.56	HECToR
2048	36.26	25.26	9.04	20.85	46.62	Jugene
4096	41.36	14.09	8.81	21.18	63.82	HECToR
4096	35.73	21.78	8.33	21.78	44.88	Jugene
8192	37.10	10.21	6.69	16.30	61.05	HECToR
8192	33.01	24.35	8.19	19.30	45.89	Jugene

Table IV. Detailed report of the cost for the main steps in a time loop for *Incompact3d*, for a $2048 \times 2048 \times 2048$ simulations with 2048 computational cores (2 and 4 cores over 4) but with a different mapping on HECToR.

Map	Cores	Conv-Diff. (%)	Poisson (%)	GradP (%)	Div(∗2) (%)	Com (%)	s/step
1×2048	2/4	31.46	41.00	7.47	18.18	72.51	18.01
1×2048	4/4	37.72	30.47	8.88	21.78	68.95	26.63
2×1024	2/4	32.06	39.76	7.30	19.79	71.30	17.23
2×1024	4/4	37.65	28.79	8.70	23.64	65.87	25.20
4×512	2/4	33.03	37.18	8.30	20.18	67.24	17.36
4×512	4/4	39.16	26.00	9.53	24.10	62.40	23.94
8×256	2/4	34.94	33.16	8.98	20.85	65.38	15.81
8×256	4/4	41.23	22.31	9.77	25.47	64.01	23.42
16×128	2/4	36.83	30.28	9.55	21.48	63.36	15.93
16×128	4/4	42.14	21.55	10.66	24.35	59.10	22.07
32×64	2/4	38.33	27.93	10.18	22.18	63.72	14.09
32×64	4/4	43.39	19.90	11.26	24.18	61.74	21.83
64×32	2/4	38.14	27.84	10.39	22.84	60.06	14.55
64×32	4/4	42.73	20.27	11.54	24.19	57.82	22.72
128×16	2/4	38.72	26.13	11.12	22.77	59.47	15.54
128×16	4/4	43.01	19.54	11.59	24.71	59.09	22.77
256×8	2/4	39.34	24.82	11.21	23.38	55.92	16.95
256×8	4/4	42.77	19.05	11.51	25.44	51.09	23.15
512×4	2/4	42.62	17.71	13.11	25.53	49.59	23.61
512×4	4/4	45.34	13.96	14.08	25.80	51.86	30.91
1024×2	2/4	45.12	11.46	15.62	26.93	37.52	36.70
1024×2	4/4	45.84	9.85	16.14	27.49	37.34	47.76
2048×1	2/4	44.90	8.15	15.91	30.67	36.89	56.09
2048×1	4/4	46.21	6.91	16.74	29.70	35.56	74.09

for a simulation with 201 million mesh nodes on 128 computational cores. Unfortunately, we tried to use Scalasca with more than 8192 computational cores in order to investigate the behaviour of the code with tens of thousands of computational cores but it was not working properly with *Incompact3d*, the profiling software seems not mature enough for such extreme conditions.

We have also decided to investigate in detail the influence of the 2D mapping on the communication time which is also compared to the wall clock time. The problem size ($2048 \times 2048 \times 2048$ mesh nodes) and the number of computational cores (2048) are fixed. The results are presented in Table IV. The main feature here is that there is no clear link between the wall clock time and

DOI: 10.1002/fld

the communication cost. The wall clock time is strongly influenced by the shape of the pencil that impacts the computation time. Indeed, for each pencil, and because sixth-order finite difference schemes are used, the computation consists in performing operations in 3D loops. Therefore, the size and shape of the inner loop direction directly impact the cost of a simulation. For instance, for the 2048×1 mapping, even if less than 36.89% is spent in communication, the wall clock time is very important, more than three times by comparison with the fastest one. Again, it proves that it is not possible to deduce the performance of the code just with the communication cost. It seems that the fastest mapping, $P_{\text{row}} \times P_{\text{col}} = 32 \times 64$ (with 2 and 4 cores per processor), is obtained when the communication part counts for about 60% of the total wall clock time. The balance between communication and calculation seems optimum when $P_{\text{row}} \simeq P_{\text{col}}$, when the 2D shape of the pencil is close to a square, i.e. $n_y/P_{\text{row}} = n_z/P_{\text{col}}$ for all the X-pencils, $n_x/P_{\text{row}} = n_z/P_{\text{col}}$ for all the Y-pencils and $n_x/P_{\text{row}} = n_y/P_{\text{col}}$ for all the Z-pencils. Another interesting feature here is the decreasing cost of the Poisson equation when increasing P_{row}. Note that the Poisson equation is just a division in spectral space so that its cost is mainly due to the communications and intensive algorithm of the FFT library. Finally, it is important to recap here that, despite our optimized pencil arrangement, no less than 55 (up to 67, for different sets of boundary conditions) global transpose operations are needed at each time step, without affecting too much the scaling of the code when thousands of computational cores are used.

Note also that for a $4096 \times 4096 \times 2048$ mesh nodes simulation using 4096 computational cores on HECToR, we have observed a performance of 0.285 Gflops per computational core for a total of 1171 Gflops. There are many factors in the computer performance other than raw floating-point computation speed, such as access performance, interprocessor communications, cache coherence and memory hierarchy. For these reasons, a numerical code is only capable of a small fraction of the 'theoretical peak' of a computational core. For the present simulation, 3.1% of the 'theoretical peak' of the computational core are reached.

5.6. *Practical flow*: *multiscale-generated turbulence*

From a physical point of view, *Incompact3d* will help us to understand the recent experiments performed at Imperial College London on fractal generated turbulence: The Turbulence, Mixing and Flow Control group has been working for more than 8 years on new flow concepts. One very recent example of a new flow concept originating from this group is that generated by multiscale (fractal) objects. Multiscale (fractal) is a geometrical concept in which a given pattern (cross, I or square in Figure 12) is repeated and split into parts, each of which is a reduced copy of the whole. A multiscale (fractal) object can be designed to be immersed in any fluid flow where there is a need to control the mixing and/or the noise generated by the turbulent flow. Many wind tunnel measurements have been performed with impressive results [37–39]. It was found that, unlike a regular object (where the turbulence is generated by only one scale), a slight modification of one of the object's parameters can deeply modify the turbulence generated by

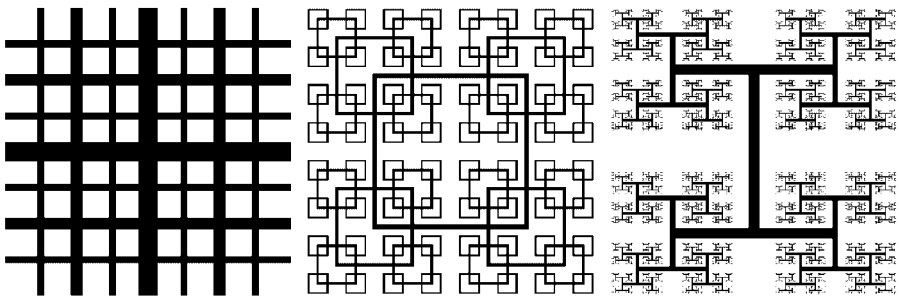

Figure 12. Scaled diagrams of a fractal cross grid (left), a fractal square grid (middle) and a fractal I grid (right).

DOI: 10.1002/fld

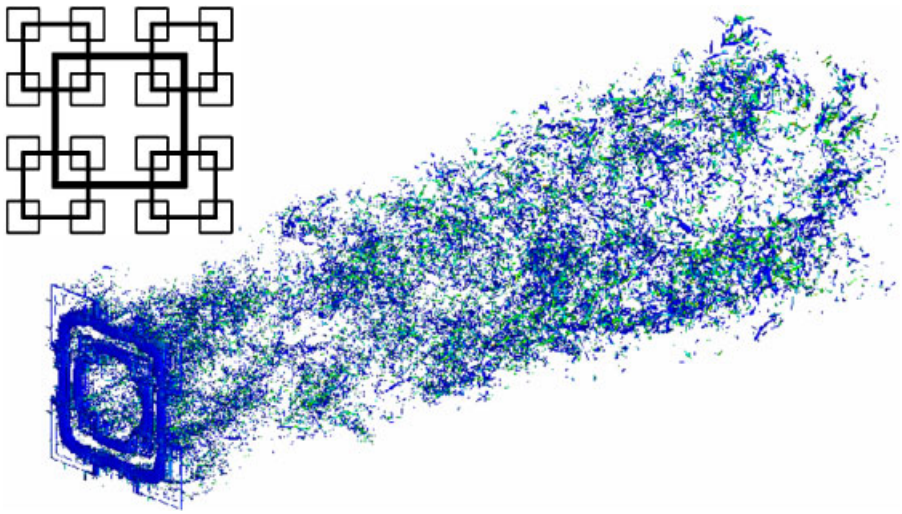

Figure 13. 3D Enstrophy isosurfaces of the turbulent flow generated by a fractal square grid with three fractal iterations (see top left) with the new version of the code. 3456 computational cores were used for this simulation.

the fluid's impact on the object. Furthermore, the experiments exhibit one disconcerting property unique to such flows: it is possible to decouple the input energy from the levels of turbulence downstream of the multiscale object. These novel objects offer possibilities for new flow solutions pertaining to industrial mixers, silent airbrakes, new ventilation and combustion devices. In order to better understand the origins of the original properties of multiscale objects, it is necessary to undertake high-fidelity simulations of such unique complex flows. Because of the complexity of the flow configuration, these simulations require hundreds of millions mesh nodes and therefore it was almost impossible three years ago to perform the numerical counterpart of the experimental measurements.

Several simulations of multiscale-generated flows mainly based on a square pattern with only a limited number of multiscale iterations (3) and with a relatively small input velocity by comparisons with experiments have already been performed with success with the old version of the code [4, 13, 14, 22]. Even if the preliminary results are very encouraging (some quantitative agreements have been found with the experimental measurements), there is a clear need for bigger simulations with the new version of *Incompact3d* that can only be undertaken with a decent wall clock time. For instance, we are currently performing multiscale-generated turbulence simulations with 768 million mesh nodes (see Figure 13 for a 3D visualization of the flow). With the old version of the code, such simulations would have required about 530 h on 288 computational cores (about 150 000 core hours) in order to get well-converged data on time. With the new version of code, we can run the same simulations with 3456 computational cores on HECToR for just 36 h (about 125 000 core hours) to have the same level of convergence. For instance, the fully turbulent flow presented in Figure 13 can now be obtained in about 12 h on HECToR with 3456 computational cores. This time corresponds to a transient stage to evacuate the initial conditions and to a period where the flow is becoming fully turbulent. With the old version of the code, it would have taken about 130 h to obtain the same turbulent state.

Pioneering research on fractal-generated turbulent flows is particularly important because the properties of some of them (exponential rather than power-law turbulence decay and kinetic energy dissipation rate inversely proportional to Reynolds number) are so different from the usual properties of idealized homogeneous isotropic turbulence. We are offered with a unique opportunity to exploit these differences for an unprecedented attempt at understanding the turbulence problem, using HPC.

DOI: 10.1002/fld

6. CONCLUSION

A numerical strategy suited for massive parallel platforms in order to run high-resolution DNS of incompressible flows on thousands of computational cores is presented in this paper. The approach proposed is based on a combination of high-order compact schemes, IBM and a 2D domain decomposition. *A priori*, the combination of high-order schemes with domain decomposition can be problematic because of the implicit nature of the schemes. However, the parallelization of *Incompact3d* has been successfully performed and the code is able to solve computationally very large fluid-flow problems with good efficiency on a large range of massive parallel platforms. Portability has been successfully realized using generic independent-platform tools and protocols like MPI. The global transformation operations are quite expensive with up to 90% of the total cost of a simulation. However, we have shown that the scalability remains acceptable up to 262 144 computational cores. We have also shown that there is no simple link between the scaling of the code and the time spent in communication. We also exhibit the importance of the shape of the 2D MPI process grid $P_{\text{row}} \times P_{\text{col}}$ that can impact directly on the time spent in the calculations and therefore influence the wall clock time of a simulation.

In order to improve the performance of the code, especially on the next generation of processors that will be based on a large number of cores, the potential of implementing OpenMP (Open Multi-Processing), which is a shared memory parallel programming protocol will be investigated. OpenMP can provide a second level of parallelism for improved performance on massive parallel platforms having multi-core processors. We also would like to improve the management by the code of the different cache levels of each computational core. The cache is a smaller, faster memory which stores copies of the data from the most frequently used main memory locations. Nowadays, there are three levels of caches per computational core. One interesting feature on the new massive parallel platforms is that the third level of cache memory can be shared by all the computational cores of a processor. For instance, in the new CRAY XT6 system which is based on 6-core processors with 4 processors per node, the 6 computational cores of a processor are sharing 6 MB of memory. We are investigating the possibility of taking advantage of this specificity in order to improve the performance of the communication in the code, even if it will reduce the portability of the code.

ACKNOWLEDGEMENTS

Sylvain Laizet acknowledges support from the EPSRC grant EP/E029515/1 and the UK Turbulence consortium (EP/G069581/1) for the CPU time made available to us on HECToR without which this study would not have been possible. He also thanks Eric Boyer, Prof. Eric Lamballais and Prof. Christos Vassilicos for very useful discussions.

Ning Li thank colleagues Chris Armstrong and Ian Bush at NAG for their significant help in developing the 2D decomposition library code.

The work on JADE has been performed under the HPC-EUROPA2 project (project number: 228398) with the support of the European Commission—Capacities Area—Research Infrastructures).

For the simulations on Jaguar, this research used resources of the National Center for Computational Sciences at Oak Ridge National Laboratory, which is supported by the Office of Science of the Department of Energy under Contract DE-AC05-00OR22725.

Sylvain Laizet and Ning Li acknowledge that the developments outlined in this paper have been achieved with the assistance of high performance computing resources (Tier-0) provided by PRACE on Jugene based in Germany.

REFERENCES

1. Canuto C, Hussaini MY, Quarteroni A, Zang TA. *Spectral Methods in Fluid Dynamics*. Springer: New York, 1988.
2. Ishihara T, Gotoh T, Kaneda Y. Study of high Reynolds number isotropic turbulence by direct numerical simulation. *Annual Review of Fluid Mechanics* 2009; **41**:165–180.
3. Deville MO, Fischer PF, Mund EH. *High-order Methods for Incompressible Fluid Flow*. Cambridge University Press: Cambridge, 2002.

DOI: 10.1002/fld

4. Laizet S, Lamballais E, Vassilicos JC. A numerical strategy to combine high-order schemes, complex geometry and parallel computing for high resolution DNS of fractal generated turbulence. *Computers and Fluids* 2010; **39**(3):471–484.
5. Laizet S, Lamballais E. High-order compact schemes for incompressible flows: a simple and efficient method with the quasi-spectral accuracy. *Journal of Computational Physics* 2009; **228**(16):5989–6015.
6. Avital EJ, Sandham ND, Luo KH. Stretched Cartesian grids for solution of the incompressible Navier–Stokes equations. *International Journal for Numerical Methods in Fluids* 2000; **33**:897–918.
7. Cain AB, Ferziger JH, Reynolds WC. Discrete orthogonal function expansions for non-uniform grids using the fast Fourier transform. *Journal of Computational Physics* 1984; **56**:272–286.
8. Kim J, Moin P. Application of a fractional-step method to incompressible Navier–Stokes equations. *Journal of Computational Physics* 1985; **59**:308–323.
9. Swarztrauber PN. The methods of cyclic reduction, Fourier analysis and the FACR algorithm for the discrete solution of Poisson's equation on a rectangle. *SIAM Review* 1977; **19**:490–501.
10. Wilhelmson RB, Ericksen JH. Direct solutions for Poisson's equation in three dimensions. *Journal of Computational Physics* 1977; **25**:319–331.
11. Chen L, Coleman SW, Vassilicos JC, Hu Z. Acceleration in turbulent channel flow. *Journal of Turbulence* 2010; **11**(41):1–23.
12. Dallas V, Vassilicos JC, Hewitt GF. Strong polymer-turbulence interactions in viscoelastic turbulent channel flow. *Physica Review E* 2010; in press.
13. Laizet S, Vassilicos JC. Direct numerical simulation of turbulent flows generated by regular and fractal grids using an immersed boundary method. *Proceedings of TSFP 6*, Seoul, 2009.
14. Laizet S, Vassilicos JC. Multiscale generation of turbulence. *Journal of Multiscale Modelling* 2009; **1**:177–192.
15. Lamballais E, Silvestrini J, Laizet S. Direct numerical simulation of a separation bubble on a rounded finite-width leading edge. *International Journal of Heat and Fluid Flow* 2008; **29**:612–625.
16. Lamballais E, Silvestrini J, Laizet S. Direct numerical simulation of flow separation behind a rounded leading edge: study of curvature effects. *International Journal of Heat and Fluid Flow* 2010; DOI: 10.1016/j.ijheatfluidflow.2009.12.007.
17. Laurendeau E, Jordan P, Bonnet JP, Delville J, Parnaudeau P, Lamballais E. Subsonic jet noise reduction by fluidic control: the interaction region and the global effect. *Physics of Fluids* 2008; **20**(10):101519.
18. Lele SK. Compact finite difference schemes with spectral-like resolution. *Journal of Computational Physics* 1992; **103**:16–42.
19. Parnaudeau P, Carlier J, Heitz D, Lamballais E. Experimental and numerical studies of the flow over a circular cylinder at Reynolds number 3900. *Physics of Fluids* 2008; **20**:085101.
20. Parnaudeau P, Lamballais E, Heitz D, Silvestrini JH. Combination of the immersed boundary method with compact schemes for DNS of flows in complex geometry. *Proceedings of DLES-5*, Munich, 2003.
21. FFTW official website. Available from: http://www.fftw.org.
22. Laizet S, Vassilicos JC. Direct numerical simulation of fractal-generated turbulence. *Proceedings of DLES-7*, Trieste, 2008.
23. Top500 official website. Available from: http://top500.org.
24. Donzis DA, Yeung PK, Pekurovsky D. Turbulence simulations on $o(10^4)$ processors. *Tera Grid Conference*, 2008. Available from: http://www.sdsc.edu/us/resources/p3dfft/.
25. P3D FFT official website. Available from: http://www.sdsc.edu/us/resources/p3dfft/.
26. Mercier P, Deville M. A multidimensional compact high-order scheme for 3-D Poisson's equation. *Journal of Computational Physics* 1981; **39**:443–455.
27. ACML official website. Available from: developer.amd.com/cpu/Libraries/acml/pages/default.aspx.
28. Source code of Plimpton's FFT library. Available from: http://www.sandia.gov/sjplimp/docs/fft/README.html.
29. Takahashi's FFTE Fast Fourier Transform package. Available from: http://www.ffte.jp/.
30. Li N, Laizet S. 2DECOMPFFT a highly scalable 2d decomposition library and FFT interface. *Cray User Group 2010*, Edinburgh, 2010.
31. Source code of Glassman's algorithm in Fortran. Available from: http://www.jjj.de/fft/glassman-fft.f.
32. FFTPACK Fortran package. Available from: http://www.netlib.org/fftpack/.
33. Intel MKL library official website. Available from: www.intel.com/cd/ids/developer/asmo-na/eng/223902.htm?page=1.
34. IBM ESSL official website. Available from: www-03.ibm.com/systems/software/essl/index.html.
35. Official website for CrayPat. Available from: http://docs.cray.com/books/S-2376-41/S-2376-41.pdf.
36. Scalasca official website. Available from: http://www.scalasca.org.
37. Hurst D, Vassilicos JC. Scalings and decay of fractal-generated turbulence. *Physics of Fluids* 2007; **19**:035103.
38. Seoud RE, Vassilicos JC. Dissipation and decay of fractal-generated turbulence. *Physics of Fluids* 2007; **19**:105108.
39. Mazellier N, Vassilicos JC. Turbulence without Richardson–Kolmogorov cascade. *Physics of Fluids* 2010; **22**:075101.

DOI: 10.1002/fld